DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	EP-A-0 395 348 (APPLE COMPUTER, INC.) 31 October 1990 * the whole document *	1-21	G06F9/302 G06F7/50 G06F7/52
X	GB-A-2 215 496 (TEXAS INSTRUMENTS LIMITED) 20 September 1989 * the whole document *	1-4,6,7, 9,10	
X	EP-A-0 239 899 (KABUSHIKI KAISHI TOSHIBA) 7 October 1987 * the whole document *	11-21	
X	GB-A-2 172 129 (RAYTHEON CO.) 10 September 1986 * the whole document *	1-4,6,7, 9,10	
P,A	EP-A-0 577 483 (SGS-THOMSON) 5 January 1994 * the whole document *	5,8	

	TECHNICAL FIELDS SEARCHED (Int.Cl.6)
	G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	14 March 1995	Daskalakis, T

FIGURE 11

FIGURE 10

321

D0 B1 D0 B0

304 303 302 301

0   0   D0B1 D0B0

308 307 306 305

D1 B1 D1 B0

0   0   D1B1 D1B0

C0 A1 C0 A0

312 311 310 309

C2A3 C2A2 0   0

C1 A1 C1 A0

316 315 314 313

C3A3 C3A2 0   0

FIGURE 9

320

C1A1 C1A0   C0A1 C0A0   0   0   0   D1B1   D1B0 D0B1 D0B0

0   0   0   0   0   0

PARTIAL
PRODUCT
SUM

E3   E2   E1   E0   F3   F2   F1   F0

321

Y0 X3 Y0 X2 Y0 X1 Y0 X0

304 303 302 301

Y0X3 Y0X2 Y0X1 Y0X0

Y1 X3 Y1 X2 Y1 X1 Y1 X0

308 307 306 305

Y1X3 Y1X2 Y1X1 Y1X0

Y2 X3 Y2 X2 Y2 X1 Y2 X0

312 311 310 309

Y2X3 Y2X2 Y2X1 Y2X0

Y3 X3 Y3 X2 Y3 X1 Y3 X0

316 315 314 313

Y3X3 Y3X2 Y3X1 Y3X0

FIGURE 8

320

Y3X3	Y3X2	Y2X3 Y2X2		Y3X0	Y1X2	Y1X1			
		Y3X1	Y1X3	Y2X1	Y0X3	Y0X2	Y1X0 Y0X1	Y0X0
				Y2X0			

PARTIAL PRODUCT SUM

Z7 Z6 Z5 Z4 Z3 Z2 Z1 Z0

YO X3   YO X2   YO X1   YO X0

204   203   202   201

YOX3   YOX2   YOX1   YOX0

Y1 X3   Y1 X2   Y1 X1   Y1 X0

208   207   206   205

Y1X3   Y1X2   Y1X1   Y1X0

Y2 X3   Y2 X2   Y2 X1   Y2 X0

212   211   210   209

Y2X3   Y2X2   Y2X1   Y2X0

Y3 X3   Y3 X2   Y3 X1   Y3 X0

FIGURE 7
(PRIOR ART)

216   215   214   213

Y3X3   Y3X2   Y3X1   Y3X0

Y2X3 Y2X2   Y1X2   Y1X1

Y3X3 Y3X2   Y3X1 Y1X3 Y3X0 YOX3 Y1X0 YOX1 YOX0
                    Y2X1 Y2X0 YOX2
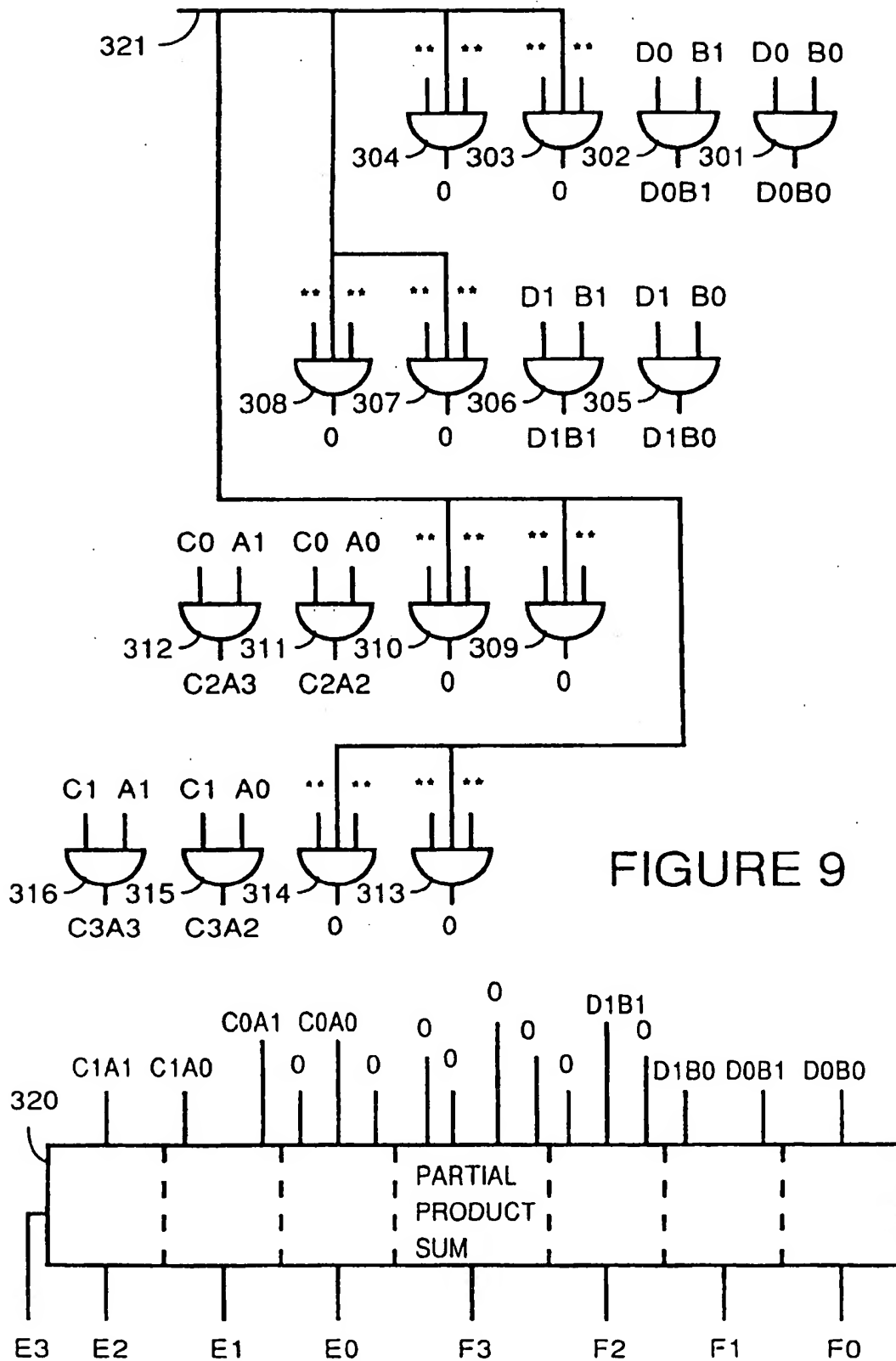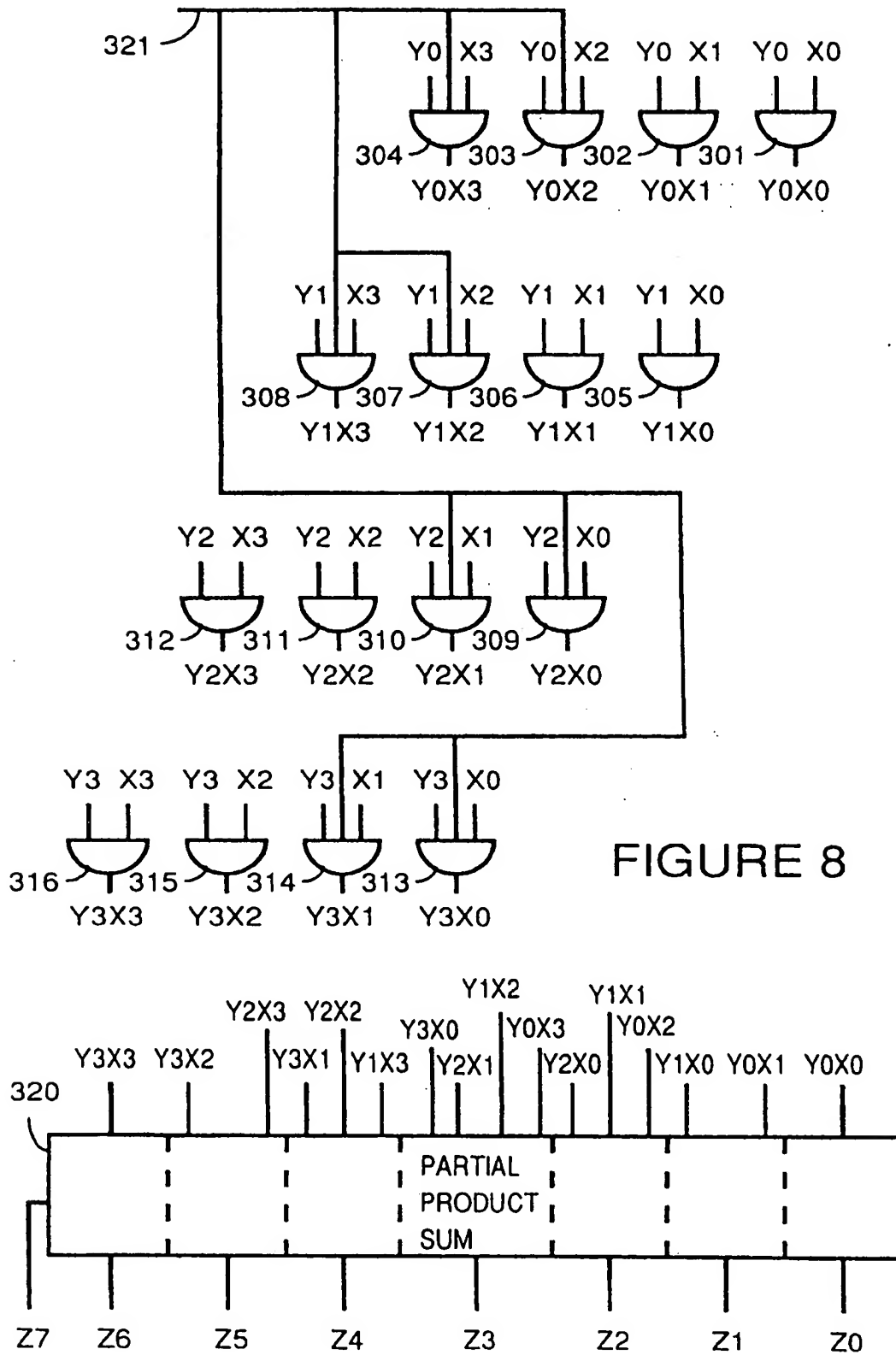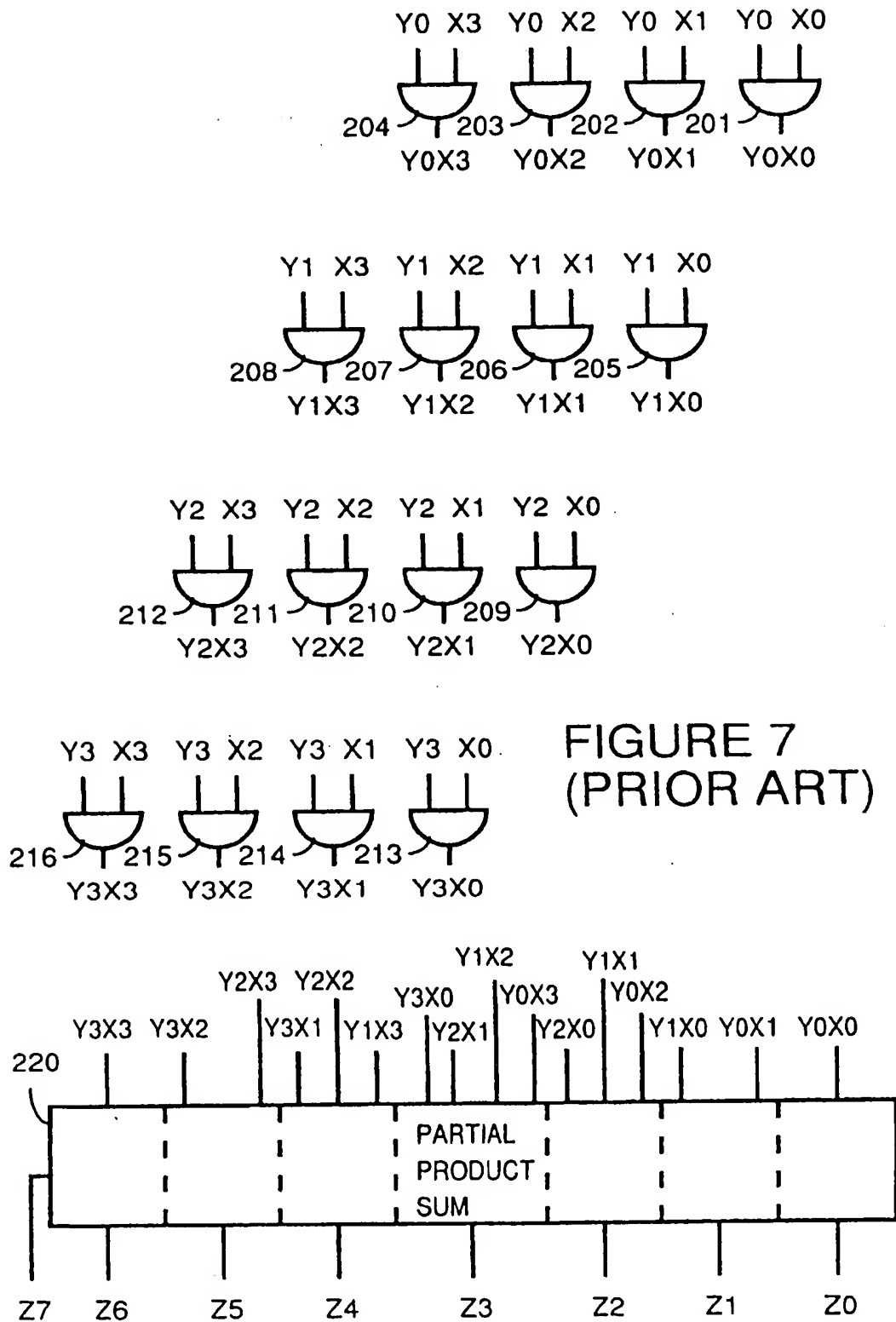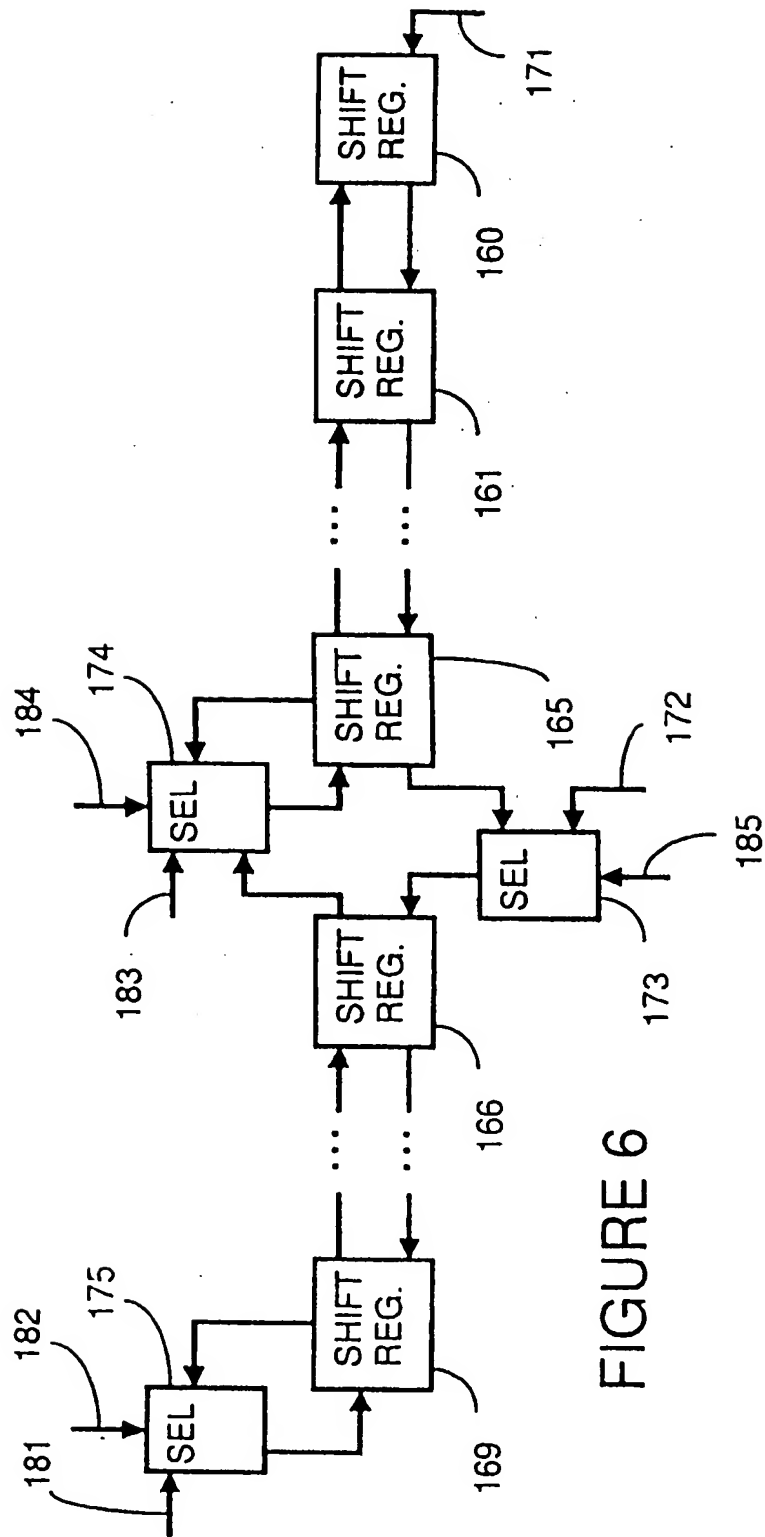
220

PARTIAL
PRODUCT
SUM

Z7   Z6   Z5   Z4   Z3   Z2   Z1   Z0

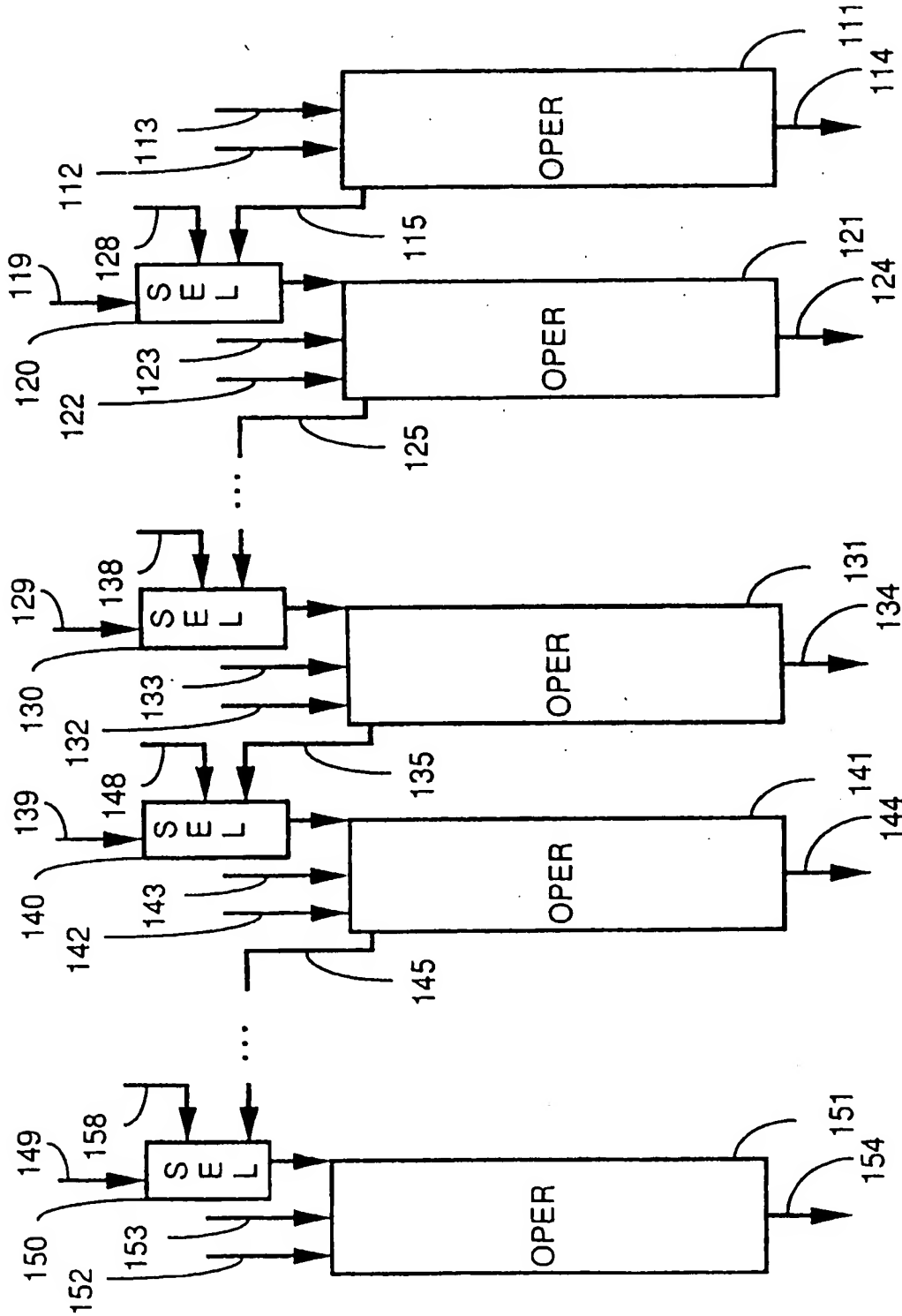FIGURE 6

FIGURE 5

FIGURE 4

FIGURE 3

LOW ORDER BIT
OPERATIONS

41

43

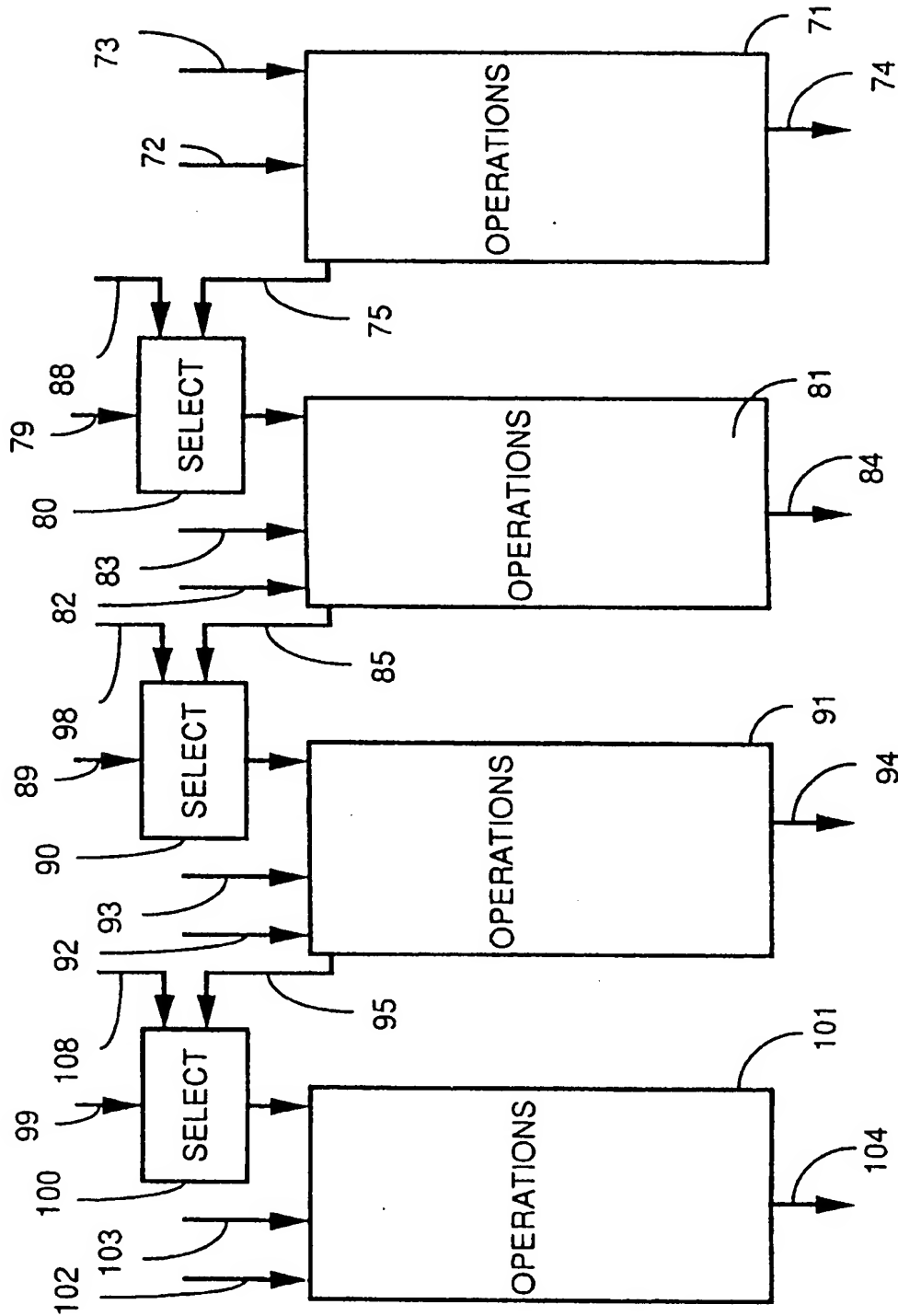42

59

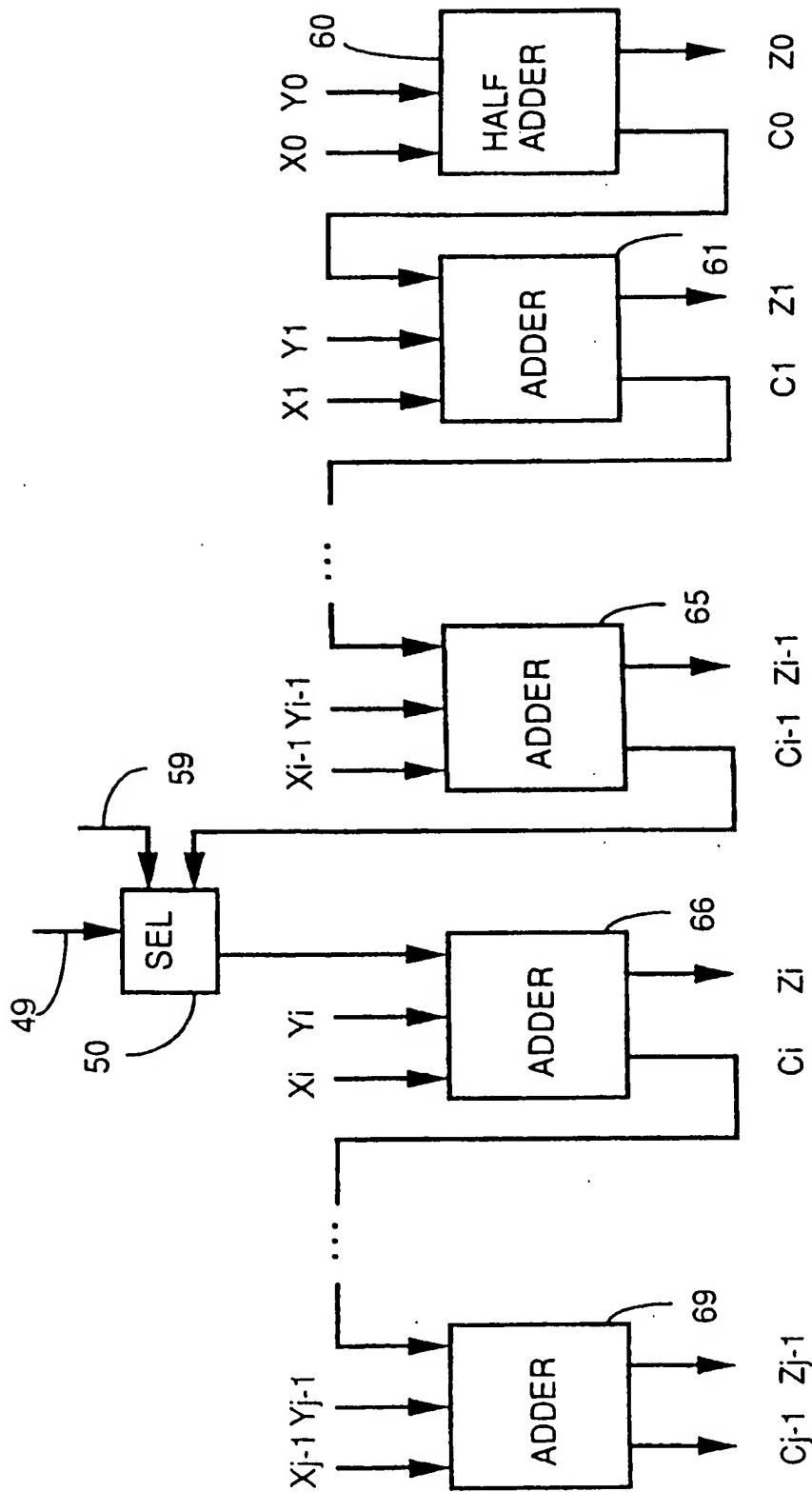45

44

SEL

49
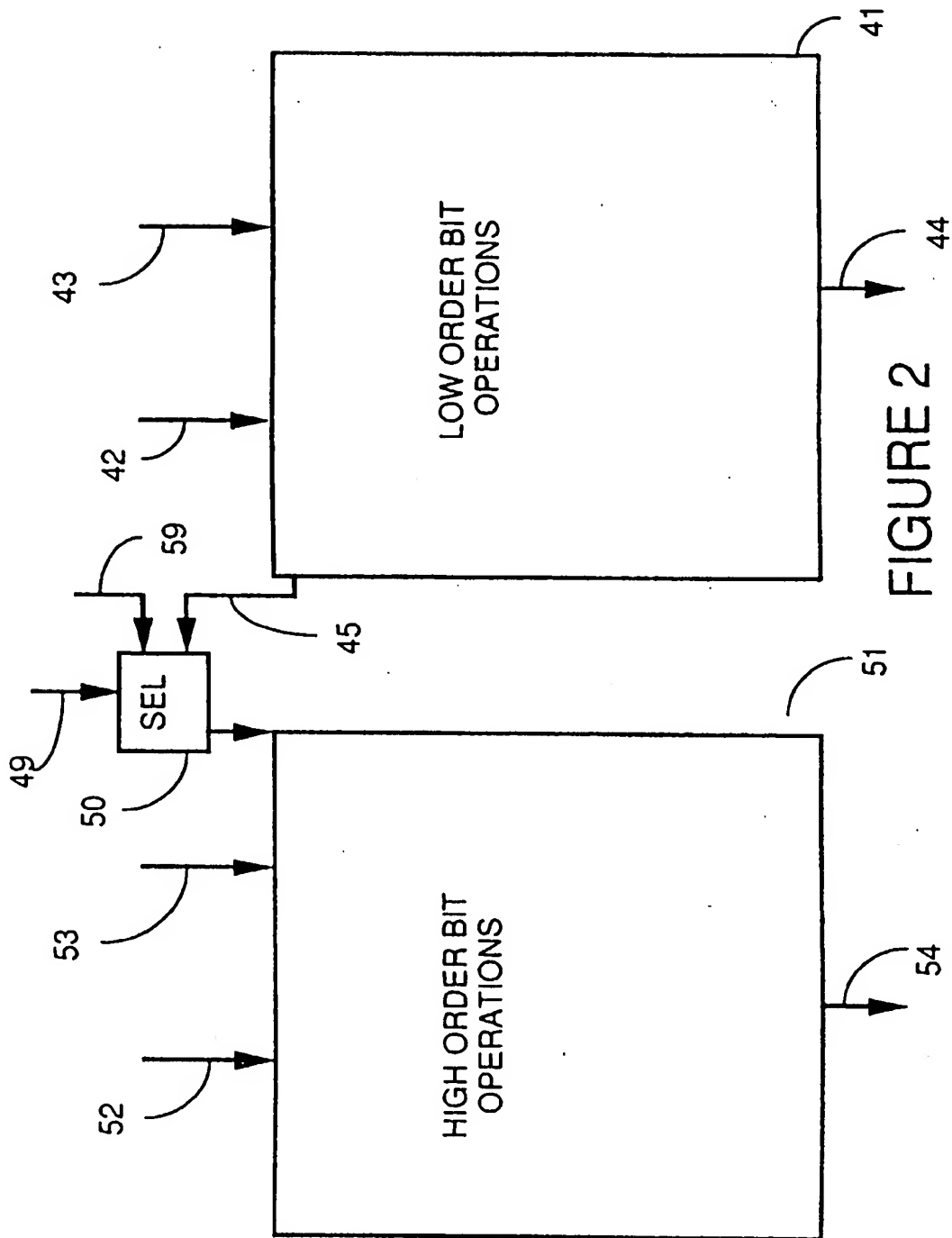
50

51

53

52

HIGH ORDER BIT
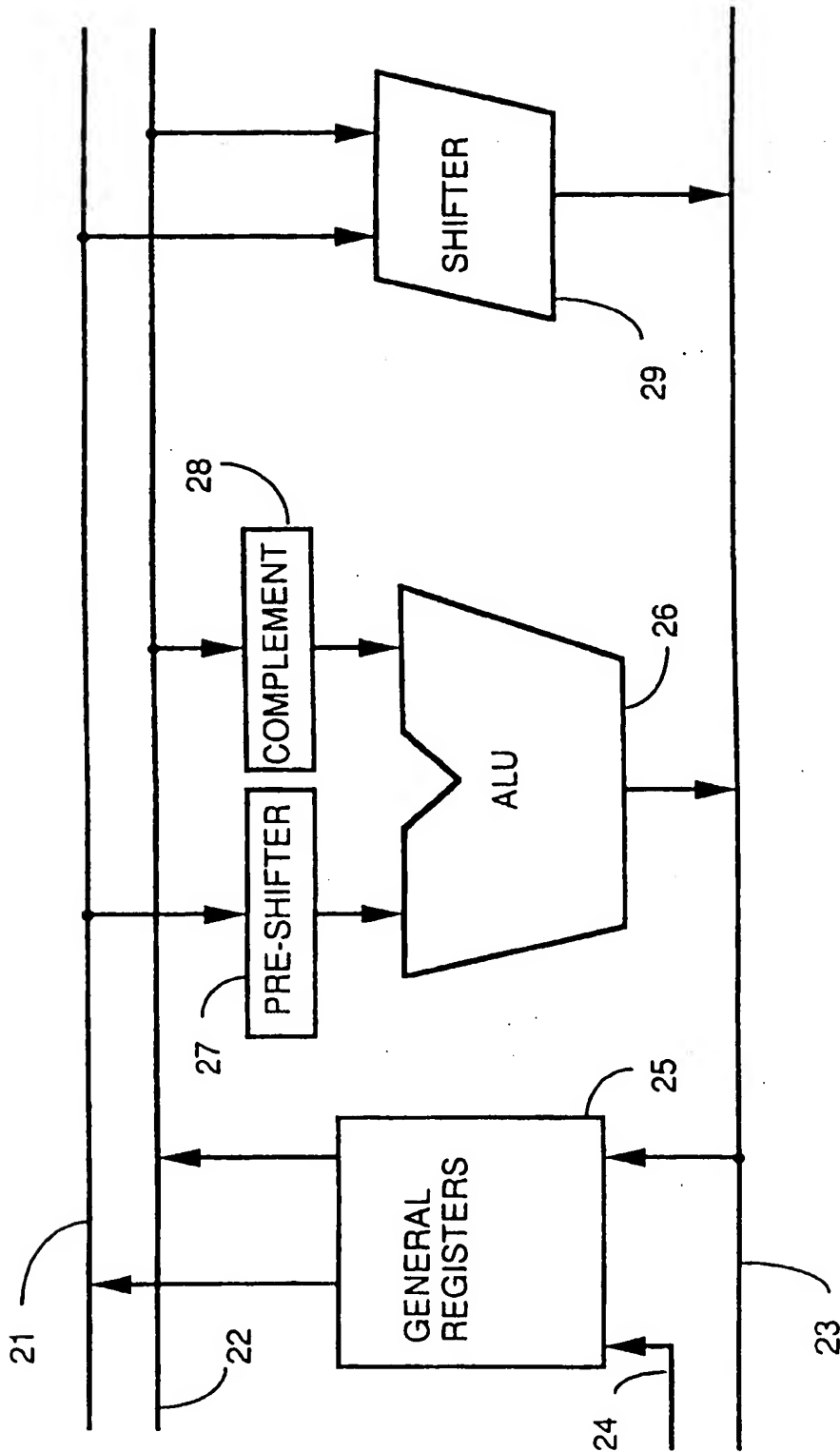OPERATIONS

54

FIGURE 2

FIGURE 1

when performing parallel addition on operands. .

11. A multiplier which both implements whole word multiplication and implements parallel multiplication of sub-word multiplicands, the multiplier comprising:

    partial product generation means (301-316) for generating partial products;

    partial product sum circuitry (320), coupled to the partial product generation means (301-316), for summing the partial products to produce a result;

    selection means (321) for selecting one of whole word multiplication and parallel multiplication of sub-word multiplicands; and,

    partial product selection means, coupled to the partial product generation means (301-316) and to the selection means (321), for in response to the selection means (321) selecting parallel multiplication of sub-word multiplicands forcing selected partial products to have a new value, thereby implementing parallel multiplication of sub-word multiplicands.

12. A multiplier as in claim 11 wherein the partial product selection means, in response to the selection means (321) selecting parallel multiplication of sub-word multiplicands, forces the selected partial products to have a value of 0.

13. A multiplier as in claim 12 wherein the partial product generation means (301-316) comprises an array of logic AND gates (301-316), each logic AND gate in the array of logic AND gates (301-316) generating a partial product.

14. A multiplier as in claim 13 wherein the partial product selection means comprises third inputs to at least a portion of the logic AND gates (301-316).

15. A multiplier as in claim 12 wherein when the multiplier is implementing whole word multiplication, the partial product selection means does not force any partial products to have a value of 0.

16. A multiplier as in claim 11 wherein the multiplier is a Booth-encoded multiplier.

17. A method for performing both multiplication of whole word multiplicands and parallel multiplication of sub-word multiplicands using a single hardware multiplier, the method comprising the steps of:

    (a) generating partial products;

    (b) in response to a selection to perform parallel multiplication of sub-word multiplicands, forcing selected partial products to have a new value; and,

    (c) summing the partial products to produce a result, the summing performed using partial product sum circuitry (320)

18. A method as in claim 17 wherein step (b) includes in response to the selection to perform parallel multiplication of sub-word multiplicands, forcing selected partial products to have a value of 0.

19. A method as in claim 18 wherein step (a) is performed using an array of logic AND gates (301-316), each logic AND gate in the array of logic AND gates (301-316) generating a partial product.

20. A method as in claim 19 wherein in step (b) forcing selected partial products to have a value of 0 is implemented by placing a logic 0 on inputs to a portion of the logic AND gates (301-316).

21. A method as in claim 18 wherein in step (b) when the multiplier is implementing whole word multiplication, not forcing any partial products to have a value of 0.

16

(51,81,121) to the third partition circuitry (91,131) when performing parallel operations on operands; and,

third selection means, coupled between the third partition circuitry (91,131) and the fourth partition circuitry (101,141), for allowing data to propagate from the third partition circuitry (91,131) to the fourth partition circuitry (101,141) when performing operations on full word length operands, and for allowing prevention of data from propagating from the third partition circuitry (91,131) to the fourth partition circuitry (101,141) when performing parallel operations on operands.

4. A functional unit as in claim 1 wherein the first selection means (50,80,120) includes means for forwarding a logic 0 to the second partition circuitry (51,81,121) when performing parallel additions on operands with bit lengths which are smaller than a bit length of the full word length operands, and for forwarding a logic 1 to the second partition circuitry (51,81,121) when performing parallel subtractions on operands with bit lengths which are smaller than a bit length of the full word length operands.

5. A functional unit as in claim 1 wherein the functional unit comprises a carry look-ahead adder (60,61,65,66,69).

6. A method for providing for parallel data processing within a single processor, the method comprising the steps of:

(a) performing operations on a first set of bits from at least one operand in first partition circuitry (41,71,111);

(b) performing operations on a second set of bits from the at least one operand in second partition circuitry (51,81,121);

(c) when performing operations on full word length operands allowing data from the first partition circuitry (41,71,111) to effect the calculation of results by the second partition circuitry (51,81,121); and,

(d) when performing parallel operations on operands, preventing data from the first partition circuitry (41,71,111) to effect the calculation of results by the second partition circuitry (51,81,121).

7. A method as in claim 6 wherein

step (a) includes performing an addition operation on low order bits of the plurality of operands;

step (b) includes an addition operation on high order bits of the plurality of operands;

step (c) includes allowing a carry from the first partition circuitry (41,71,111) to effect the calculation of results by the second partition circuitry (51,81,121) when performing the addition on full word length operands; and,

step (d) includes preventing a carry from the first partition circuitry (41,71,111) to effect the calculation of results by the second partition circuitry (51,81,121) when performing parallel addition on operands.

8. A method as in claim 6 wherein

step (a) includes performing a carry look-ahead addition operation on low order bits of the plurality of operands; and,

step (b) includes performing a carry look-ahead addition operation on high order bits of the plurality of operands.

9. A method as in claim 6 wherein

step (a) includes performing a subtraction operation on low order bits of the plurality of operands; and,

step (b) includes performing a subtraction operation on high order bits of the plurality of operands.

10. A method as in claim 6 wherein

step (a) includes performing a carry-propagate addition operation on low order bits of the plurality of operands;

step (b) includes a carry-propagate addition operation on high order bits of the plurality of operands;

step (c) includes allowing a carry from propagating to the second partition circuitry (51,81,121) when performing the addition on full word length operands; and,

step (d) includes preventing a carry from propagating to the second partition circuitry (51,81,121)

15

Kaufmann, 1990, Appendix, pp. A-39 through A-49. As, in the case of the multiplier above, the value of some partial product terms generated by the Booth-encoded multiplier are changed to take into account the parallel processing, as will be understood by those skilled in the art.

More specifically, for a Booth-encoded multiplier, the AND gates 301 through 316 shown in Figures 8
5 and 9 are replaced by multiplexors. For example, a Booth-encoded multiplier using the "overlapping triplets" method examines three bits of the multiplier (i.e., y multiplicand) each time, instead of one bit each time, to generate a row of partial products that is one of +x, +2x, -2x, -x or zero, instead of a row of partial products which is always +x or 0 as in the multiplier shown in Figures 8 and 9. This may be implemented as a five-to-one multiplexor. The name "overlapping triplets" is due to the fact that this method looks at
10 three bits of the multiplier (y multiplicand) and retires two bits of the multiplier (y multiplicand) for each row. The overlapping occurs when, for the next row, the least significant bit of the three multiplier (y multiplicand) bits used by this next row was the most significant bit of the three multiplier bits used from the previous row.

To implement parallel sub-word multiplication, the bits of the x multiplicand that do not correspond to
15 the sub-word product whose partial product rows are being formed are set to zero. This can be implemented with multiplexors like in the unmodified Booth-encoded multiplier, modifying the control signals to the multiplexors. The sign of the partial product row may also be used as an additional input to the multiplexors.

The foregoing discussion discloses and describes merely exemplary methods and embodiments of the
20 present invention. As will be understood by those familiar with the art, the invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. Accordingly, the disclosure of the present invention is intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the following claims.

25 **Claims**

1. A functional unit within a processing system, the functional unit comprising:
    first partition circuitry (41,71,111) which performs operations on a first set of bits from a plurality of operands;
30   second partition circuitry (51,81,121) which performs operations on a second set of bits from the plurality of operands; and,
    first selection means (50,80,120), coupled between the first partition circuitry (41,71,111) and the second partition circuitry (51,81,121), for allowing data to propagate from the first partition circuitry (41,71,111) to the second partition circuitry (51,81,121) when performing operations on full word length
35   operands, and for preventing data from propagating from the first partition circuitry (41,71,111) to the second partition circuitry (51,81,121) when performing parallel operations on operands with bit lengths which are smaller than a bit length of the full word length operands.

2. A functional unit as in claim 1 wherein
40   the first partition circuitry (41,71,111) performs an addition operation on low order bits of the plurality of operands,
    the second partition circuitry (51,81,121) performs the addition operation on high order bits of the plurality of operands, and
    the first selection means (50,80,120) is a selector which allows a carry to propagate from the first
45   partition circuitry (41,71,111) to the second partition circuitry (51,81,121) when performing addition of full word length operands, and which prevents the carry from propagating from the first partition circuitry (41,71,111) to the second partition circuitry (51,81,121) when performing parallel additions on sub-word length operands.

50 3. A functional unit as in claim 1 additionally comprising:
    third partition circuitry (91,131) which performs operations on a third set of bits from a plurality of operands;
    fourth partition circuitry (101,141) which performs operations on a fourth set of bits from the plurality of operands;
55   second selection means (90,130), coupled between the second partition circuitry (51,81,121) and the third partition circuitry (91,131), for allowing data to propagate from the second partition circuitry (51,81,121) to the third partition circuitry (91,131) when performing operations on full word length operands, and for allowing prevention of data from propagating from the second partition circuitry

14

Table 11

```
AAAAAAAABBBBCCCC
000000000000iiii        F
000000000000iiii        F
000000000000iiii        F
000000000000iiii        F
00000000hhhh0000        E
00000000hhhh0000        E
00000000hhhh0000        E
00000000hhhh0000        E
gggggggg00000000        D
gggggggg00000000        D
gggggggg00000000        D
gggggggg00000000        D
gggggggg00000000        D
gggggggg00000000        D
gggggggg00000000        D
gggggggg00000000        D
GGGGGGGGGGGGGGGGGHHHHHHHHIIIIIIII
```

As can be seen from Table 11 above, in a first parallel multiplication, an eight-bit multiplicand AAAAAAAA$_{(base\ 2)}$ is multiplied by an eight-bit multiplicand DDDDDDDD$_{(base\ 2)}$ to produce a sixteen-bit result GGGGGGGGGGGGGGGG$_{(base\ 2)}$. In a second parallel multiplication, a four-bit multiplicand BBBB$_{(base\ 2)}$ is multiplied by a four-bit multiplicand EEEE$_{(base\ 2)}$ to produce an eight-bit result HHHHHHHH$_{(base\ 2)}$. In a third parallel multiplication a four-bit multiplicand CCCC$_{(base\ 2)}$ is multiplied by a four-bit multiplicand FFFF$_{(base\ 2)}$ to produce an eight-bit result IIIIIIII$_{(base\ 2)}$. As will be understood by those skilled in the art, for every partial product shown in Table 11 with a value of zero, it is necessary to have a three input logic AND gate or its logic equivalent to allow the partial product to be forced to zero when performing parallel multiplication operations. However, when a mixture of different sized partitions are done, as in Table 11, in some implementations different control inputs may be needed to force different partial product terms to zero, as will be understood by those of ordinary skill in the art.

As may be understood from the above-discussion, by selectively forcing partial products of a multiplier to zero parallel multiplication of partial words may be filly implemented in a multiplier. The size of the word, the number of parallel multiplications simultaneously performed and the size of the partial words may be freely varied in accordance with the teaching of the present invention.

Figure 11 shows an example of instructions that can be executed in accordance with the preferred embodiment of the present invention. For example, instruction 500 includes a field 501, a subfield 502 of field 501 a field 503 a field 504 and a field 505. Field 501 sets out the op code. Field 501 sets out, for example an add a shift and add, a subtract, a shift and subtract, a shift left, a shift right, a multiply, or any number of other operations Subfield 502 of 501 indicates whether the operation is to be performed as parallel operations, and if so, what is the size of the operands. Field 503 indicates a first source register. Field 504 indicates a second source register. Field 505 indicates a destination register.

As will be understood in the art, instruction 500 illustrates one of many possible ways an instruction can be organized. For example, instruction 510 shows an alternate embodiment in that the parallel operation indication is in a separate field. Specifically, instruction 510 includes a field 511, a field 512, a field 513, a field 514 and a field 515. Field 511 sets out the op code. Field 511 sets out, for example, an add, a shift and add, a subtract, a shift and subtract, a shift left, a shift right, a multiply, or any number of other operations. Field 512 indicates whether the operation is to be performed as parallel operations, and if so, what is the size of the operands. Field 513 indicates a first source register. Field 514 indicates a second source register. Field 515 indicates a destination register.

As will be understood in the art, the present invention also works for other multipliers where partial products are generated. For example, the present invention also may be utilized in a Booth-encoded multiplier. In a Booth-encoded multiplier, fewer rows of partial product terms are generated by considering more than one bit of the multiplier (y-multiplicand) for each row of the partial product term. See for example, John Hennessy & David Patterson, Computer Architecture. A Quantitative Approach, Morgan

Table 10

```
AAAAAAAABBBBBBBB
  00000000ffffffff      D
  00000000ffffffff      D
  00000000ffffffff      D
  00000000ffffffff      D
  00000000ffffffff      D
  00000000ffffffff      D
  00000000ffffffff      D
  00000000ffffffff      D
  eeeeeeee00000000      C
  eeeeeeee00000000      C
  eeeeeeee00000000      C
  eeeeeeee00000000      C
  eeeeeeee00000000      C
  eeeeeeee00000000      C
  eeeeeeee00000000      C
  eeeeeeee00000000      C
EEEEEEEEEEEEEEEEFFFFFFFFFFFFFFFF
```

As can be seen from Table 10 above, in a first parallel multiplication, an eight-bit multiplicand $AAAAAAAA_{(base\ 2)}$ is multiplied by an eight-bit multiplicand $CCCCCCCC_{(base\ 2)}$ to produce a sixteen-bit result $EEEEEEEEEEEEEEEE_{(base\ 2)}$. In a second parallel multiplication, an eight-bit multiplicand $BBBBBBBB_{(base\ 2)}$ is multiplied by an eight-bit multiplicand $DDDDDDDD_{(base\ 2)}$ to produce a sixteen-bit result $FFFFFFFFFFFFFFFF_{(base\ 2)}$. Multiplication of two whole word (sixteen-bit) multiplicands is implemented by the multiplier by not forcing any of the partial products to zero.

While the above description has showed parallel multiplication of half words, it will be understood by persons of ordinary skill in the art that both the number of parallel multiplications performed and the size of the partial word may be varied by selecting the appropriate partial products to force to zero.

For example, the sixteen-bit multiplier implemented as described by Table 4 (and/or Table 10) may be utilized to perform three simultaneous parallel multiplications by providing circuitry, such as shown in Figure 8 and Figure 9, to force partial products to zero in accordance with the teaching of the present invention. Thus, modifying the multiplier described by Table 4 in accordance with the teaching of the present invention allows, for example, the performance of one parallel multiplication using eight-bit multiplicands and two parallel multiplication using four-bit multiplicands as implemented by Table 11 below:

Table 8

		A	A	B	B	
		0	0	£	£	D
	0	0	£	£		D
	e	e	0	0		C
	e	e	0	0		C
E	E	E	E	F	F	F	F

As illustrated by Table 7 and Table 8, parallel multiplication of partial words is implemented in a multiplier by forcing selected partial products in the multiplier to zero. In general, a standard multiplier of any size may be utilized to perform parallel multiplication by forcing unused partial products to zero. The partial products are forced to logic 0, for example, by using one or more control inputs and three input logic AND gates (or their equivalents).

For example, as discussed above, an eight-bit multiplier may be implemented as described by Table 3. This same multiplier may be utilized to perform parallel multiplication of partial word multiplicands by providing circuitry, such as shown in Figure 8 and Figure 9, to force partial products to zero in accordance with the teaching of the present invention. No modification is necessary to the partial product sum circuitry. Thus, modifying the multiplier described by Table 3 in accordance with the teaching of the present invention allows, for example, the performance of two parallel multiplications using four-bit multiplicands as implemented by Table 9 below:

Table 9

				A	A	A	A	B	B	B	B	
				0	0	0	0	£	£	£	£	D
			0	0	0	0	£	£	£	£		D
		0	0	0	0	£	£	£	£			D
	0	0	0	0	£	£	£	£				D
	e	e	e	e	0	0	0	0				C
	e	e	e	e	0	0	0	0				C
	e	e	e	e	0	0	0	0				C
	e	e	e	e	0	0	0	0				C
E	E	E	E	E	E	E	E	F	F	F	F	F	F	F	F

As can be seen from Table 9 above, in a first parallel multiplication of partial word multiplicands, a four-bit multiplicand $AAAA_{(base\ 2)}$ is multiplied by a four-bit multiplicand $CCCC_{(base\ 2)}$ to produce an eight-bit result $EEEEEEEE_{(base\ 2)}$. In a second parallel multiplication of partial word multiplicands, a four-bit multiplicand $BBBB_{(base\ 2)}$ is multiplied by a four-bit multiplicand $DDDD_{(base\ 2)}$ to produce an eight-bit result $FFFFFFFF_{(base\ 2)}$. Multiplication of two whole word (eight-bit) multiplicands is implemented by the multiplier by not forcing any of the partial products to zero.

Likewise, as discussed above, a sixteen-bit multiplier may be implemented as shown by Table 4. This same multiplier may be utilized to perform parallel multiplications of partial word multiplicands by providing circuitry, such as shown in Figure 8 and Figure 9, to force partial products to zero in accordance with the teaching of the present invention. No modification is necessary to the partial product sum circuitry. Thus, modifying the multiplier described by Table 4 in accordance with the teaching of the present invention allows, for example, the performance of two parallel multiplications using eight-bit (partial word) multiplicands as implemented by Table 10 below:

11

Table 5

				X_3	X_2	X_1	X_0	
				Y_0X_3	Y_0X_2	Y_0X_1	Y_0X_0	Y_0
			Y_1X_3	Y_1X_2	Y_1X_1	Y_1X_0		Y_1
		Y_2X_3	Y_2X_2	Y_2X_1	Y_2X_0			Y_2
	X_3X_3	X_3X_2	X_3X_1	X_3X_0				Y_3
Z_7	Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0	

A comparison of Table 5 with Table 1 above, confirms that when line 321 is set to logic 1, operation of the multiplier shown in Figure 8 is identical to operation of the multiplier shown in Figure 7. Therefore, similar to Table 2 above, the simplified notation may be used to describe operation of the multiplier shown in Figure 8 as in Table 6 below:

Table 6

				X	X	X	X	
				z	z	z	z	Y
			z	z	z	z		Y
		z	z	z	z			Y
	z	z	z	z				Y
z	z	z	z	z	z	z	z	

Figure 9 shows the multiplier shown in Figure 8, except however that control line 321 is set at logic 0. This forces half the partial products to zero allowing the multiplier to perform parallel multiplication of partial (two-bit) words. That is, in a first multiplication, a two-bit multiplicand A_1A_0 (base 2) is multiplied by a two-bit multiplicand C_1C_0 (base 2) to produce a four-bit result $E_3E_2E_1E_0$ (base 2). In a second multiplication, a two-bit multiplicand B_1B_0 (base 2) is multiplied by a two-bit multiplicand D_1D_0 (base 2) to produce a four-bit result $F_3F_2F_1F_0$ (base 2). The partial products not used for the parallel multiplications are forced to logic zero. The parallel multiplication may be represented in table form as shown in Table 7 below:

Table 7

				A_1	A_0	B_1	B_0	
				0	0	D_0B_1	D_0B_0	D_0
			0	0	D_1B_1	D_1B_0		D_1
	C_0A_1	C_0A_0	0	0				C_0
C_1A_1	C_1A_0	0	0					C_1
E_3	E_2	E_1	E_0	F_3	F_2	F_1	F_0	

Using the simplified notation first introduced in Table 2, the multiplier shown in Figure 9 may be represented as in Table 8 below:

10

The multiplier shown in Table 3 multiplies an eight-bit first multiplicand $XXXXXXXX_{(base\ 2)}$ with an eight-bit second multiplicand $YYYYYYYY_{(base\ 2)}$ to produce an sixteen-bit result $ZZZZZZZZZZZZZZZZ_{(base\ 2)}$.

Similarly, using the simpler notation of Table 2 and Table 3 (but eliminating spaces between bit positions) a sixteen-bit multiplier may be described as shown in Table 4 below:

Table 4

<u>XXXXXXXXXXXXXXXX</u>

ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y .
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y
ZZZZZZZZZZZZZZZZ	Y

ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ

The multiplier shown in Table 4 multiplies a sixteen-bit first multiplicand $XXXXXXXXXXXXXXXX_{(base\ 2)}$ with a sixteen-bit second multiplicand $YYYYYYYYYYYYYYYY_{(base\ 2)}$ to produce a thirty-two-bit result $ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ_{(base\ 2)}$.

In accordance with preferred embodiments of the present invention, a standard multiplier may be modified to implement a multiplier which provides parallel multiplication of partial words in addition to multiplication of whole words. For example, Figure 8 shows a four-bit multiplier in accordance with the preferred embodiment of the present invention. Logic AND gates 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315 and 316 generate partial products for the multiplication. A partial product sum circuit 320 sums the partial products generated by logic AND gates 301 through 316 to produce the result.

In the multiplier shown in Figure 8, partial product sum circuit 320 may be implemented exactly the same as partial product sum circuit 220 shown in Figure 7. The difference between the multiplier shown in Figure 8 and the multiplier shown in Figure 7, is the addition of a control line 321, which is connected to an additional input included in each of logic AND gates 303, 304, 307, 308, 309, 310, 313 and 314.

As shown in Figure 8, when control line 321 is set at logic 1, the multiplier performs a whole word multiplication on a four-bit first multiplicand $X_3X_2X_1X_0$ (base 2) and a four-bit second multiplicand $Y_3Y_2Y_1Y_0$ (base 2) to produce an eight-bit result $Z_7Z_6Z_5Z_4Z_3Z_2Z_1Z_0$ (base 2). The two multiplicands, $X_3X_2X_1X_0$ and $Y_3Y_2Y_1Y_0$, the partial products generated by logic AND gates 301 through 316, and the result produced by partial product sum circuit 320 may be represented in table form as shown in Table 5 below:

For example, Figure 7 shows a four-bit multiplier in accordance with the prior art. The multiplier multiplies a four-bit first multiplicand $X_3X_2X_1X_0$ (base 2) with a four-bit second multiplicand $Y_3Y_2Y_1Y_0$ (base 2) to produce an eight-bit result $Z_7Z_6Z_5Z_4Z_3Z_2Z_1Z_0$ (base 2). As is understood by those skilled in the art, logic AND gates 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215 and 216 may be used to generate partial products for the multiplication. A partial product sum circuit 220 sums the partial products generated by logic AND gates 201 through 216 to produce the result.

The two multiplicands, $X_3X_2X_1X_0$ and $Y_3Y_2Y_1Y_0$, the partial products generated by logic AND gates 201 through 216, and the result produced by partial product sum circuit 220 may be placed in a table in such a way as to summarize operation of the multiplier. For example, such a table is shown as Table 1 below:

Table 1

```
                              X3     X2     X1     X0
                            Y0X3   Y0X2   Y0X1   Y0X0              Y0
                     Y1X3   Y1X2   Y1X1   Y1X0                     Y1
              Y2X3   Y2X2   Y2X1   Y2X0                            Y2
       X3X3   Y3X2   Y3X1   Y3X0                                   Y3
  Z7     Z6     Z5     Z4     Z3     Z2     Z1     Z0
```

In the notation used in Table 1 above, the bit position of each bit of both multiplicands and the result is specifically identified. Additionally, the bits of the multiplicand which are used to form each partial product are specifically set out. As is understood by those skilled in the art, the information shown in Table 1 above may be set out using abbreviated or simplified notation, as in Table 2 below:

Table 2

```
                    X  X  X  X
                    z  z  z  z          Y
                 z  z  z  z             Y
              z  z  z  z                Y
           z  z  z  z                   Y
        Z  Z  Z  Z  Z  Z  Z  Z
```

In Table 2 above, each bit of the first multiplicand is represented by an "X", each bit of the second multiplicand is represented by a "Y", each bit of a partial product is represented by a "z", and each bit of the result is represented by a "Z". Using the simpler notation of Table 2, an eight-bit multiplier may be described as shown in Table 3 below:

Table 3

```
                          X  X  X  X  X  X  X  X
                          z  z  z  z  z  z  z  z              Y
                       z  z  z  z  z  z  z  z                 Y
                    z  z  z  z  z  z  z  z                    Y
                 z  z  z  z  z  z  z  z                        Y
              z  z  z  z  z  z  z  z                           Y
           z  z  z  z  z  z  z  z                              Y
        z  z  z  z  z  z  z  z                                 Y
     z  z  z  z  z  z  z  z                                    Y
  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z  Z
```

Equation 5

$$Gs[i] = (M\,[i] * F) + (!M\,[i] * (A[i] * B[i]))$$

$$= (M\,[i] * F) + (!M\,[i] * G[i])$$

$$Ps[i] = Pm[i]$$

Now if M[i] is 1, the value of Gs[i] is determined by F. If M [i] is 0 the value of Gs[i] is determined by A[i] and B[i] as it was previously. The propagate does not have to be forced by the F signal.

The equation of the carry out is given by Equation 6 below:

Equation 6 $C[i] = Gs\,[i] + Ps\,[i] * C[i-1]$

As will be understood by persons of skill in the art, principles of the present invention are not confined to arithmetic operations within computer system ALUs. For example, partitioning as shown in the ALU may also be extended to other entities within the computer system which operate on data. For example, Figure 6 shows the present invention embodied in pre-shifter 27. The same embodiment of the present invention may also be used to implement shifter 29. Partitioning of pre-shifter 27 and shifter 29 allows, for example, for the implementation of parallel shift-and-add operations and parallel shift operations

Pre-shifter 27 is shown to include a shift register one-bit slice 160, a shift register one-bit slice 161, a shift register one-bit slice 165, a shift register one-bit slice 166 and a shift register one-bit slice 169.

When data is shifted to the left, a datum on input 171, typically a logic 0 value, is used as input to shift register one-bit slice 160. When data is shifted to the right, a selector 175 in response to a control input 182 selects either a datum on input 181 (a logic 0 value or a logic 1 value) or selects the value currently stored by shift register one-bit slice 169 to be input to shift register one-bit slice 169.

Wherever the shifter is to be partitioned, additional selectors are added to the shifter. For example, Figure 6 shows the shifter partitioned between shift register one-bit slice 165 and shift register one-bit slice 166. There a selector 174 and a selector 173 have been added. For shift operations on partitioned operands, when data is shifted to the left, selector 173, in response to a control input 185, selects a datum on input 172, typically a logic 0 value, to be used as input to shift register one-bit slice 166. For shift operations on full word operands, when data is shifted to the left, selector 173 selects output from shift register one-bit slice 165 to be used as input to shift register one-bit slice 166.

For shift operations on partitioned operands, when data is shifted to the right, selector 174 in response to a control input 184 selects either a datum on input 183 (a logic 0 value or a logic 1 value) or selects the value currently stored by shift register one-bit slice 166 to be input to shift register one-bit slice 165. For shift operations on full word operands, when data is shifted to the right, selector 174 selects output from shift register one-bit slice 166 to be used as input to shift register one-bit slice 165.

Figure 6 shows a shifter with only two partitions. As will be understood from the foregoing discussion of partitions in an ALU, the shifter can be partitioned in a variety of ways. For example, a 64-bit shifter may be partitioned into two, four, eight, sixteen, thirty-two or sixty-four bit equal size partitions. Additionally, it is not a requirement of the present invention that partitions each operate on equal number of bits.

Wile the above embodiment describes the pre-shifter 27 and shifter 29 implemented as a shift register consisting of a series of one bit slices, alternative preferred embodiments are pre-shifters and shifters implemented with multiplexors. Typically, pre-shifter 27 is implemented by a one level of multiplexors, since it can usually shift by at most a small number of bits, for example, 0, 1, 2, 3 or 4 bits. Shifter 29 is typically implemented by three levels of multiplexors, where each level of multiplexor is a four-to-one multiplexor. For example, in a 64-bit shifter 29, the first level of multiplexors will shift either 0, 16, 32 or 48 bits. The second level of multiplexors can shift either 0, 4, 8 or 12 bits. The third level of multiplexors can shift 0, 1, 2 or 3 bits. This gives a shift of any number of bits from 0 to 63. In such a shifter built up of 3 stages of multiplexors, one-bit slices can still be identified. However the blocking of the shifts between any two bits may need to be done in one or more of the three multiplexer stages, as will be understood by those of ordinary skill in the art.

Principles of the present invention may also be extended to other elements in a computer system. For example, a multiplier may be implemented in accordance with a preferred embodiment of the present invention to allow for partial word parallel multiplications in addition to whole word multiplications.

7

bit Z_1. A full adder 465 receives a single bit X_{i-1} of the first operand, a single bit Y_{i-1} of the second operand and a carry bit C_{i-2}. Full adder 465 produces a sum bit Z_{i-1}. A full adder 466 receives a single bit X_i of the first operand, a single bit Y_i of the second operand and a carry bit C_{i-1}. Full adder 466 produces a sum bit Z_i. A full adder 469 receives a single bit X_{j-1} of the first operand, a single bit Y_{j-1} of the second operand and a carry bit C_{j-2}. Full adder 469 produces a sum bit Z_{j-1}.

In the embodiment of the adder shown in Figure 10, "j" is the size of the data path and the bit length of full word operations. Also, "i" is equal to "j" divided by 2. For example, "j" is equal to 32 and "i" is equal to 16. Alternately, when j is equal to 32, i may be equal to any integer less than 32.

When performing operations using "j"-bit full word operands, an enable bit 452 is equal to logic one and allows all carries to propagate. When performing two parallel operations using "i"-bit sub-word operands partitioned between bits i and i + 1, enable bit 452 is equal to logic zero and prevents the carry propagating across the partition boundary. Instead the value on line 451 is used as the value forwarded to full adder 466. When an "add" is being performed, a logic 0 is placed on input line 451. When a "subtract" is being performed, a logic 1 is placed on input line 451.

Operation of carry look-ahead adders are well understood in the art. For example, suppose A[i] is one bit of an input, B[i] is one bit of the other input and S[i] is one bit of the sum from the adder. Then, the sum from one bit of the adder is given by Equation 1 below:

Equation 1 $S[i] = A[i] \text{ XOR } B[i] \text{ XOR } C[i-1]$

In equation 1, C[i-1] is the carry out of the previous bits of the carry look-ahead adder. The carry look-ahead adder works on generating these carry bits quickly.

Let G[i] be a signal which signifies that a carry is to be generated by this bit and P[i] be a signal that a carry may propagate from the previous bits to the output of this bit. These are determined in accordance with Equation 2 below:

Equation 2 $G[i] = A[i] \text{ AND } B[i]; \quad P[i] = A[i] \text{ OR } B[i];$

Therefore, for four bits within a carry look-ahead adder, the carry bits may be generated as in Equation 3 below:

Equation 3

$$
\begin{aligned}
C[i] \quad &= \quad G[i] + P[i] * (G[i\text{-}1] + P[i\text{-}1] * (G[i\text{-}2] + P[i\text{-}2] * (G[i\text{-}3] + P[i\text{-}3] * C[i\text{-}4]))) \\
C[i\text{-}1] &= \qquad\qquad\quad G[i\text{-}1] + P[i\text{-}1] * (G[i\text{-}2] + P[i\text{-}2] * (G[i\text{-}3] + P[i\text{-}3] * C[i\text{-}4])) \\
C[i\text{-}2] &= \qquad\qquad\qquad\qquad\qquad G[i\text{-}2] + P[i\text{-}2] * (G[i\text{-}3] + P[i\text{-}3] * C[i\text{-}4]) \\
C[i\text{-}3] &= \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad G[i\text{-}3] + P[i\text{-}3] * C[i\text{-}4]
\end{aligned}
$$

In equation 3 above, "*" is equivalent to a logic AND operation and "+" is equivalent to a logic OR operation.

When implementing a preferred embodiment of the present invention, a carry is stopped at a particular bit if the Generate G[i] and Propagate P[i] are forced to be false. For instance, in equation 3 above, if G[i-3] and P[i-3] are false, C[i-3] will be false and C[i-4] can never effect the value of C[i-2], C[i-1], and C[i]. Likewise, if G[i-2] and P[i-2] are false, C[i-2] will be false and G[i-3] and P[i-3] and C[i-4] can never effect the value of C[i-1] and C[i].

If we let M [i] be a mask bit that breaks the carry-chain between bit [i] and bit [i + 1] when M [i] is 1, then a new Equation 4 can be generated as follows:

Equation 4 $Gm[i] = !M[i] * (A[i] * B[i]) \quad Pm[i] = !M[i] * (A[i] + B[i])$

Now if M [i] is 1, a carry will not be allowed to be generated from bit [i] or to propagate through bit [i].

For subtraction by creating the one's complement of one of the operands and adding it to the other operand with a carry in (two's complement arithmetic), a carry must be forced to be generated in a bit when M[i] is 1.

Let F be a signal that when true will force a carry to be generated in a bit when M [i] is 1. The equation for Gs[i] and Ps[i] becomes as set out in Equation 5 below:

For example, in a computer which has a sixty-four bit wide data path, each full-word operand is 64 bits. Therefore, when performing operations using 64-bit full word operands, selector 80 allows information to propagate from first partition 71 through selector 80 to second partition 81, selector 90 allows information to propagate from second partition 81 through selector 90 to third partition 91, and selector 100 allows information to propagate from third partition 91 through selector 100 to fourth partition 101. When performing two parallel operations using 32-bit half word operands, selector 80 allows information to propagate from first partition 71 through selector 80 to second partition 81, selector 90 prevents information from propagating from second partition 81 through selector 90 to third partition 91, and selector 100 allows information to propagate from third partition 91 through selector 100 to fourth partition 101. When performing four parallel operations using 16-bit quarter word operands, selector 80 prevents information from propagating from first partition 71 through selector 80 to second partition 81, selector 90 prevents information from propagating from second partition 81 through selector 90 to third partition 91, and selector 100 prevents information from propagating from third partition 91 through selector 100 to fourth partition 101.

Figure 5 shows an another alternate simplified block diagram of ALU 26 in accordance with another alternate preferred embodiment of the present invention. In Figure 5, ALU 26 is divided into partitions which are each one bit wide. A first partition 111 performs operations on a low order bit 112 of a first operand and on a low order bit 113 of a second operand to produce a low order result bit 114. A second partition 121 performs operations on a bit 122 of the first operand and a bit 123 of the second operand to produce a result bit 124. A partition 131 performs operations on a bit 132 of the first operand and a bit 133 of the second operand to produce a result bit 134. A partition 141 performs operations on a bit 142 of the first operand and a bit 143 of the second operand to produce a result bit 144. A partition 151 performs operations on a high order bit 152 of the first operand and a high order bit 153 of the second operand to produce a high order result bit 154.

In response to a control input 119, a selector 120 is used to allow information on data path 115 to propagate from first partition 111 to second partition 121 or to intercept information on data path 115 before it is propagated from first partition 111 to second partition 121. When data is intercepted the value on a line 128 is forwarded to partition 121. When an "add" is being performed, a logic 0 is placed on line 128. When a "subtract" is being performed, a logic 1 is placed on line 128.

In response to a control input 129, a selector 130 is used to allow information on a data path from an immediately prior partition (not shown) to propagate from the immediately prior partition to partition 131 or to intercept information on the data path from the immediately prior partition before it is propagated to partition 131. When data is intercepted the value on a line 138 is forwarded to partition 131. When an "add" is being performed, a logic 0 is placed on line 138. When a "subtract" is being performed, a logic 1 is placed on line 138.

In response to a control input 139, a selector 140 is used to allow information on data path 135 to propagate from partition 131 to partition 141 or to intercept information on data path 135 before it is propagated from partition 131 to partition 141. When data is intercepted the value on a line 148 is forwarded to partition 141. When an "add" is being performed, a logic 0 is placed on line 148. When a "subtract" is being performed, a logic 1 is placed on line 148.

In response to a control input 149, a selector 150 is used to allow information on a data path from an immediately prior partition (not shown) to propagate from the immediately prior partition to partition 151 or to intercept information on the data path from the immediately prior partition before it is propagated to partition 151. When data is intercepted the value on a line 158 is forwarded to partition 151. When an "add" is being performed, a logic 0 is placed on line 158. When a "subtract" is being performed, a logic 1 is placed on line 158.

The control inputs to the selectors may be used to allow parallel processing of operands of varying length. For example, in a processing system with a sixty-four bit wide data path, the control inputs could be selected so that parallel processing of two sixteen bit and four eight-bit arithmetic operations are all performed simultaneously. Additionally any bit combination which add up to no more than the word size could be used. For example, parallel processing of seventeen bit, three bit, sixteen bit, twelve bit, five bit, and eleven bit arithmetic operations can also be performed simultaneously.

The principles discussed above also apply to a carry look-ahead adder. For example, Figure 10 shows implementation of a two's complement adder with carry look-ahead within ALU 26 in accordance with another preferred embodiment of the present invention. A carry look-ahead circuit 470 produces carries for the adder. A half adder 460 receives a single bit X_0 of a first operand and a single bit Y_0 of a second operand. Half adder 460 produces a sum bit Z_0. A All adder 461 receives a single bit X_1 of the first operand, a single bit Y_1 of the second operand and a carry bit carry bit C_0. Full adder 461 produces a sum

5

Figure 3 shows implementation of a two's complement adder with carry propagate addition within ALU 26 in accordance with a preferred embodiment of the present invention. Alternately, ALU 26 includes a two's complement adder with carry look-ahead. A half adder 60 receives a single bit X_0 of a first operand and a single bit Y_0 of a second operand. Half adder 60 produces a sum bit Z_0 and a carry bit C_0. A full adder 61 receives a single bit X_1 of the first operand, a single bit Y_1 of the second operand and carry bit C_0. Full adder 61 produces a sum bit Z_1 and a carry bit C_1. A full adder 65 receives a single bit X_{i-1} of the first operand, a single bit Y_{i-1} of the second operand and a carry bit from a previous adder (i.e., C_{i-2}, not shown). Full adder 65 produces a sum bit Z_{i-1} and a carry bit C_{i-1}. A full adder 66 receives a single bit X_i of the first operand and a single bit Y_i of the second operand. Depending on a value of enable bit 49, full adder 66 also receives, through selector 50 (or equivalent logic circuitry as will be understood by persons of ordinary skill in the art), carry bit C_{i-1}. Full adder 66 produces a sum bit Z_i and a carry bit C_i. A full adder 69 receives a single bit X_{j-1} of the first operand, a single bit Y_{j-1} of the second operand and a carry bit from a previous adder (not shown). Full adder 69 produces a sum bit Z_{j-1} and a carry bit C_{j-1}.

In the embodiment of the adder shown in Figure 3, "j" is the size of the data path and the bit length of full word operations. Also, "i" is equal to "j" divided by 2. For example, "j" is equal to 32 and "i" is equal to 16.

Selector 50 is also shown in Figure 3. When performing operations using "j"-bit full word operands, enable bit 49 is equal to logic one and allows a carry to propagate through selector 50 to full adder 66. When performing two parallel operations using "i"-bit half word operands, enable bit 49 is equal to logic zero and prevents the carry from propagating through selector 50 to full adder 66. Instead the value on line 59 is forwarded to full adder 66. When an "add" is being performed, a logic 0 is placed on input line 59. When a "subtract" is being performed, a logic 1 is placed on input line 59.

While Figures 2 and 3 discuss implementations of ALU 26 with two partitions, an ALU designed in accordance with other preferred embodiments of the present invention may variously partition an ALU. For example Figure 4 shows an alternate simplified block diagram of ALU 26 in accordance with an alternate preferred embodiment of the present invention. In Figure 4, ALU 26 is divided into four partitions. A first partition 71 performs operations on low order bits 72 of a first operand and low order bits 73 of a second operand to produce low order bit results 74. A second partition 81 performs operations on bits 82 of the first operand and bits 83 of the second operand to produce result bits 84. A third partition 91 performs operations on bits 92 of the first operand and bits 93 of the second operand to produce result bits 94. A fourth partition 101 performs operations on high order bits 102 of the first operand and high order bits 103 of the second operand to produce high order bit results 104.

In response to a control input 79, a selector 80 is used to allow information on data path 75 to propagate from first partition 71 to second partition 81 or to intercept information on data path 75 before it is propagated from first partition 71 to second partition 81. Particularly, for arithmetic operations performed on full-word operands or half-word operands, information is allowed to propagate from first partition 71 through selector 80 to second partition 81. For the performance of parallel arithmetic operations on quarter-word operands selector 80 prevents information from propagating from first partition 71 to second partition 81. Instead the value on a line 88 is forwarded to partition 81. When an "add" is being performed, a logic 0 is placed on line 88. When a "subtract" is being performed, a logic 1 is placed on line 88. Generally, in logic operations there is no propagation of information between partitions.

In response to a control input 89, a selector 90 is used to allow information on data path 85 to propagate from second partition 81 to third partition 91 or to intercept information on data path 85 before it is propagated from second partition 81 to third partition 91. Particularly, for arithmetic operations performed on full-word operands, information is allowed to propagate from second partition 81 through selector 90 to third partition 91. For the performance of parallel arithmetic operations on quarter-word operands or half-word operands, selector 90 prevents information from propagating from second partition 81 to third partition 91. Instead the value on a line 98 is forwarded to partition 91. When an "add" is being performed, a logic 0 is placed on line 98. When a "subtract" is being performed, a logic 1 is placed on line 98.

In response to a control input 99, a selector 100 is used to allow information on data path 95 to propagate from third partition 91 to fourth partition 101 or to intercept information on data path 95 before it is propagated from third partition 91 to fourth partition 101. Particularly, for arithmetic operations performed on full-word operands and half-word operands, information is allowed to propagate from third partition 91 through selector 100 to fourth partition 101. For the performance of parallel arithmetic operations on quarter-word operands, selector 100 prevents information from propagating from third partition 91 to fourth partition 101. Instead the value on a line 108 is forwarded to partition 101. When an "add" is being performed, a logic 0 is placed on line 108. When a "subtract" is being performed, a logic 1 is placed on line 108.

Figure 2 shows a simplified block diagram of an arithmetic logic unit (ALU) shown in Figure 1 in accordance with a preferred embodiment of the present invention.

Figure 3 shows an implementation of a two's complement adder within the ALU shown in Figure 2 in accordance with a preferred embodiment of the present invention.

5   Figure 4 shows an alternate simplified block diagram of the arithmetic logic unit (ALU) shown in Figure 1 in accordance with an alternate preferred embodiment of the present invention.

Figure 5 shows another alternate simplified block diagram of the arithmetic logic unit (ALU) shown in Figure 1 in accordance with another alternate preferred embodiment of the present invention.

Figure 6 shows an implementation of a shifter shown in Figure 1 in accordance with a preferred
10   embodiment of the present invention.

Figure 7 shows a multiplier in accordance with the prior art.

Figure 8 and Figure 9 show a multiplier implemented in accordance with preferred embodiments of the present invention.

Figure 10 shows an implementation of a carry look-ahead adder within the ALU shown in Figure 1 in
15   accordance with an alternate preferred embodiment of the present invention.

Figure 11 shows an example of an instruction layout in accordance with an alternate preferred embodiment of the present invention.

Description of the Preferred Embodiments

20

Figure 1 shows a simplified block diagram of an operation execution data path within a processor in accordance with preferred embodiments of the present invention. Operands for upcoming operations and results from accomplished operations are stored within general registers 25. When operations are performed, a first operand stored in a first register within general registers 25 is placed on a first source bus
25   21. If the operation requires another operand, a second operand stored in a second register within general registers 25 is placed on a second source bus 22.

After performance of the operation, the result is placed on a result bus 23 and loaded into a register within general registers 25. The operation is performed by arithmetic logic unit (ALU) 26 or by a shifter 29. A pre-shifter 27 and complement circuitry 28 may each be used to modify operands before they are
30   received by ALU 26. For general background about the architecture of single processor systems constructed similarly to the present invention see, for example, Ruby B. Lee, Precision Architecture, IEEE Computer, Volume 22, No. 1, January 1989, pp. 78-91.

In accordance with the preferred embodiments of the present invention, the ALU may be partitioned to allow parallel data processing. For example, Figure 2 shows ALU 26 divided into two partitions. A first
35   partition 41 performs operations on low order bits 42 of a first operand and low order bits 43 of a second operand to produce low order bit results 44. A second partition 51 performs operations on high order bits 52 of the first operand and high order bits 53 of the second operand to produce high order bit results 54.

In response to a control input 49, a selector 50 is used to allow information on data path 45 to propagate from first partition 41 to second partition 51 or to intercept information on data path 45 before it is
40   propagated from first partition 41 to second partition 51. Particularly, for arithmetic operations performed on full-word operands, information is allowed to propagate from first partition 41 through selector 50 to second partition 51. For the performance of parallel arithmetic operations on half-word operands, selector 50 prevents information from propagating from first partition 41 to second partition 51. Generally, in logic operations, there is no propagation of information from first partition 41 to second partition 51.

45   For example, in a computer which has a thirty-two bit wide data path, each full-word operand is 32 bits. Therefore, when performing operations using 32-bit full word operands, selector 50 allows information to propagate from first partition 41 through selector 50 to second partition 51. When performing two parallel operations using 16-bit half word operands, selector 50 prevents information from propagating from first partition 41 through selector 50 to second partition 51. Instead the value on a line 59 is forwarded to
50   partition 51. When an "add" is being performed, a logic 0 is placed on input line 59. When a "subtract" is being performed, a logic 1 is placed on input line 59.

In the preferred embodiment of the present invention, a common arithmetic operation performed by ALU 26, shown in Figure 1, is two's complement addition. As is understood by those skilled in the art, the use of two's complement circuitry 28 to perform a two's complement on an operand before performing a
55   two's complement addition operation in the ALU implements a two's complement subtraction. Also, the use of pre-shifter 27 to pre-shift an operand before performing a two's complement addition operation in the ALU implements a shift and add operation.

3

Background

The present invention concerns parallel data processing in a single processor system.

In general, single processor systems sequentially perform op.rations on two operands. For example, in
5 a 32-bit computer, each integer operand is 32 bits. In a 64-bit computer, each integer operand is 64 bits.
Thus an integer "add" instruction, in a 64-bit computer, adds two 64-bit integer operands to produce a 64-
bit integer result. In most pipelined 64-bit processors, a 64-bit add instruction takes one cycle of execution
time.

In many instances the pertinent range of operands is 16 bits or less. In current 32-bit and 64-bit
10 computers, however, it still takes a full instruction to perform an operation on a pair of 16-bit operands. Thus
the number of execution cycles required to perform an operation on two 16-bit operands is the same as the
number of execution cycles required to perform the operation on two 32-bit operands in a 32-bit computer
or two 64-bit operands in a 64-bit computer.

In the prior art, parallel data processing required replicating of functional units, each functional unit able
15 to handle full word length data. See for example, Michael Flynn, Very High-Speed Computing Systems,
Proceedings of IEEE, Vol. 54, Number 12, December 1966, pp. 1901-1909. However, such implementa-
tions of parallel processing is significantly costly both in terms of hardware required and complexity in
design.

20 Summary of the Invention

In accordance with the preferred embodiment of the present invention, a system is presented which
allows parallel data processing within a single processor. In order to allow for parallel processing of data, an
arithmetic logic unit or other operation executing entity within the processing system such as a shifter is
25 partitioned. Within each partition operations are performed. When the operation is to be performed on full
word length operands, there is no parallel processing. Thus data is allowed to freely propagate across
boundaries between the partitions. When performing the operation in parallel using a plurality of less than
one full word length operands, data is prevented from being propagated across at least one boundary
between the partitions.

30 For example, when the operation is an addition operation (e.g., a two's complement addition), each of
the plurality of partitions performs an addition operation. When the addition is to be performed on full word
length operands, carries are allowed to propagate between the partitions. When performing the addition
operation in parallel on a plurality of less than one full word length operand sets, a carry is prevented from
propagating across at least one boundary between the partitions.

35 Likewise, when the operation is a shift, each of the plurality of partitions performs a shift operation.
When the shift is to be performed on full word length operands, shifts are allowed between the partitions.
When performing the operation in parallel using a plurality of less than one full word length operands, a shift
is prevented from crossing at least one boundary between the partitions.

Also in accordance with a preferred embodiment of the present invention, a multiplier implements both
40 multiplication of whole word multiplicands and parallel multiplication of sub-word multiplicands. Circuitry, for
example an array of logic AND gates (or their equivalent), generates partial products. Partial product sum
circuitry, sums the partial products to produce a result. Partial product gating means, in response to the
selection of parallel multiplication of sub-word multiplicands, forces selected partial products to have a value
of 0, thereby implementing parallel multiplication of sub-word multiplicands. When the multiplier is
45 implementing whole word multiplication, none of the partial products are forced to have a value of 0. The
partial product gating means may be implemented, for example, using third inputs to at least a portion of
the logic AND gates.

The present invention allows for a single processor system to significantly increase performance by
facilitating parallel processing operations when operands are less than the full word length. This inexpensive
50 use of parallelism results in a huge increase in performance for computations that can utilize this type of
data parallelism without significant additional cost in silicon space on a processor chip or complexity in
design. The present invention also allows for parallel processing operations performed by a processor in
response to a single instruction.

55 Brief Description of the Drawings

Figure 1 shows a simplified block diagram of an operation execution data path within a processor in
accordance with preferred embodiments of the present invention.

2

(54) Parallel data processing in a single processor.

(57) A system and a parallel data processing within a single processor. In order to allow for parallel processing of data, an arithmetic logic unit (26) or other operation executing entity within the processing system such as a shifter (29) is partitioned. Within each partition (41,51), operations are performed on a portion of one or more operands When the operation is to be performed on full word length operands, there is no parallel processing. Thus data is allowed to freely propagate across boundaries between the partitions (41,51). When performing the operation in parallel using a plurality of operands of less than one full word in length, data is prevented from being propagated across at least one boundary between the partitions (41,51). The principles of the present invention may also be utilized to implement a multiplier (301-316,320) which performs parallel multiplication of partial word multiplicands.
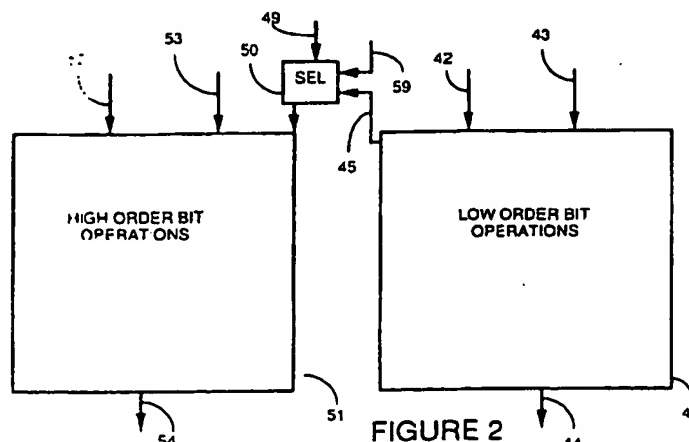
FIGURE 2